MULTI-CYCLE AT SPEED TEST

A Thesis

by

MALLIKA SHREE POKHAREL

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Duncan M. H. Walker
Committee Members,	Rabi Mahapatra
	Weiping Shi
Head of Department,	Dilma Da Silva

August 2017

Major Subject: Computer Engineering

ABSTRACT

In this research, we focus on the development of an algorithm that is used to generate a minimal number of patterns for path delay test of integrated circuits using the multi-cycle at-speed test. We test the circuits in functional mode, where multiple functional cycles follow after the test pattern scan-in operation. This approach increases the delay correlation between the scan and functional test, due to more functionally realistic power supply noise. We use multiple at-speed cycles to compact K-longest paths per gate tests, which reduces the number of scan patterns. After a path is generated, we try to place each path in the first pattern in the pattern pool. If the path does not fit due to conflicts, we attempt to place it in later functional cycles. This compaction approach retains the greedy nature of the original dynamic compaction algorithm where it will stop if the path fits into a pattern. If the path is not able to compact in any of the functional cycles of patterns in the pool, we generate a new pattern.

In this method, each path delay test is compared to at-speed patterns in the pool. The challenge is that the at-speed delay test in a given at-speed cycle must have its necessary value assignments set up in previous (preamble) cycles, and have the captured results propagated to a scan cell in the later (coda) cycles. For instance, if we consider three at-speed (capture) cycles after the scan-in operation, and if we need to place a fault in the first capture cycle, then we must generate it with two propagation cycles. In this case, we consider these propagation cycles as coda cycles, so the algorithm attempts to select the most observable path through them. Likewise, if we are placing the path test in

the second capture cycle, then we need one preamble cycle and one coda cycle, and if we are placing the path test in the third capture cycle, we require two preamble cycles with no coda cycles.

DEDICATION

To my family and everyone who believed in me

ACKNOWLEDGEMENTS

I would like to thank my M.S. advisory committee chair Dr. Duncan M. “Hank” Walker for his guidance, support and encouragement throughout my Masters life at A&M. I want to express my sincere gratitude for having faith in me and providing me constant motivation to achieve my milestones.

I am very grateful to my committee members, Dr. Rabi N. Mahapatra and Dr. Weiping Shi, for their valuable suggestions during the research. I would also want to express my sincere thankfulness to Dr. Jiang Hu for his continuous support and ideas for the research.

I want to thank all the staff, faculty and friends who have helped me in different ways during my Masters life.

Finally, I would like to specially thank my brother and family, who made it all possible to reach here with their love and support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

Part 1, faculty committee recognition

This work was supervised by a thesis committee consisting of Professor Dr. Duncan M. “Hank” Walker and Professor Dr. Rabi N. Mahapatra of the Department of Computer Science and Engineering and Professor Dr. Weiping Shi of the Department of Electrical and Computer Engineering.

Part 2, student/advisor contributions

All work for the thesis was completed by the student, under the advisement of Professor Dr. Duncan M. “Hank” Walker of the Department of Computer Science and Engineering.

Funding Sources

This work was made possible in part by the National Science Foundation under Grant Number CCF-1117982.

NOMENCLATURE

ATPG	Automatic Test Pattern Generation
DFT	Design For Test
LOC	Launch On Capture
PFT	Pseudo Functional Test
KLPG	K Longest Path Per Gate
SCOAP	Sandia Controllability/Observability Analysis Program
PSN	Power Supply Noise
PI	Primary Input
PO	Primary Output
PPI	Pseudo Primary Input
PPO	Pseudo Primary Output
FC	Fault Coverage
TF	Time Frame
NA	Necessary Assignment

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1. INTRODUCTION

1.1. Overview

The research in this thesis builds on prior research in testing the K longest paths per gate in an integrated circuit [1][2]. We extend the prior research on dynamic test compaction [3] to consider the case where each test pattern has multiple at-speed functional cycles, so the at-speed test could take place in any one of these cycles. If the test takes place in one of the interior cycles, then the at-speed cycles that precede it are treated the same as *preamble* cycles that were introduced in pseudo functional test [6], and the at-speed cycles that come after are treated the same as *coda* cycles, that were introduced to handle non-scan state elements [12]. In the coda algorithm, the most observable testable path is selected to propagate the result captured at the end of the at-speed test cycle. This coda path introduces necessary assignments (values in the circuit necessary for signal propagation) in the coda cycles.

We extend our prior compaction algorithm by considering the set of time frames associated with each test pattern (preamble, at-speed, and coda cycles). When a test is placed in a capture cycle, its necessary assignments (NAs) are placed in the corresponding time frame and subsequent coda time frames, and then checked for compatibility, first with a direct NA conflict check, and if that passes, with a satisfiability check, which produces values for the scan pattern. The algorithm is greedy in that it first attempts to place the path test in the first at-speed cycle of the first pattern. If that fails, it tries the next pattern, and so on. Then it tries the second at-speed cycle of each pattern, and so on. If

these all fail, it generates a new pattern and places the test in the first time frame. Due to memory limits, we keep only a pool of patterns in memory, and write out a pattern whenever a new pattern is created. The more path tests that can be placed in later at-speed cycles, the fewer test patterns needed. Since the extra at-speed cycles take negligible time (scan speeds are an order of magnitude slower than functional speeds), this will reduce test application time.

1.2. Delay Testing

Delay testing is used to verify the performance of a circuit with respect to its timing specifications. If the timing defect cannot be detected before the chip is "shipped", it may lead to setup or hold time violations, incorrect performance or constrained temperature or voltage range.

An error is said to have occurred in the circuit if the captured value is different from the specified value. These errors are caused by faults, which are abstractions of the physical mechanisms that cause increased circuit delay. Different fault models [13][14][15] can be used to detect faults in the circuit. In the transition fault model [24], each gate on its input or output can have a slow-to-rise (STR) or a slow-to-fall (STF) delay fault. These faults are assumed to be large enough that any path propagating through the fault site will be slow. These faults are detected by propagating a transition in the form of initializing vector (v1) and test vector (v2) which sensitizes the fault and propagates the fault to an observable PPO [16][17]. A transition fault test generator normally selects the easiest (most observable) paths as delay are accumulated in the nodes of gates. These are

often the shortest paths. Since the transition fault model targets large delay defects, the test patterns it produces may miss small delay defects.

The path delay fault model [15] targets delay accumulated along the path. Since the total number of paths is exponential in the number of gates in the circuit, we are using the K Longest Paths Per Gate (KLPG) algorithm [1][2] in this research. This algorithm was developed to test the K longest paths through each line in the circuit. It is used in finding the critical paths of the circuit. It limits the number of paths to be tested, but provides good coverage of both small local delay defects and distributed delay defects. We are focusing our research on path delay test using the KLPG algorithm.

1.3. Scan Based Test

Scan Based Test is the most used and effective testing approach in Design For Test (DFT). It consists of a set of sequential elements serially connected leading to the formation of scan chain. Each cell in a scan chain is known as Scan Flip-Flop (SFF). A SFF provide access to direct controllability (PPI: the output of a SFF) and direct observability (PPO: the input of a SFF) [20]. In our research, we are using Muxed–D Scan where a D-flip flop is converted to a Scan Flip Flop by adding a 2:1 multiplexer (MUX) at the data input as shown in Figure 1. With little modification our work can also support level sensitive scan design (LSSD).

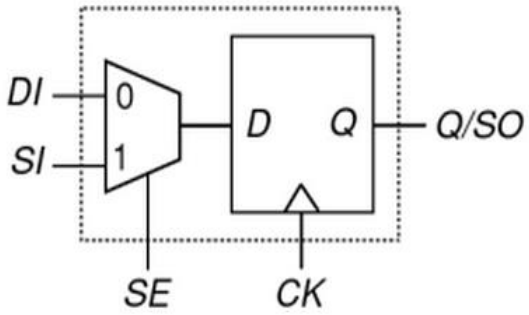

Figure 1. Muxed-D Scan [20]

The two inputs of the MUX are Data Input (DI) and Scan Input (SI) with the Select Line as Scan Enable (SE). During the scan operation, the test patterns are scanned in with the value of SE as 1. Likewise, during the normal functional operation, the data inputs are scanned as SE value is 0 for the normal operation. The Scan Output (SO) of a scan cell is connected to the SI port of the next scan cell to form a scan chain.

The different modes [20] of scan design are:

1. **Shift Mode**: During this mode, the SE signal is asserted and the scan pattern (initialization vector) is scanned in and shifted into the SFFs. The scan clock is much slower (an order of magnitude) than the functional clock.

2. **Normal mode**: In this mode, the SE signal is de-asserted and the circuit switches back to its functional operation (at-speed frequency) to launch the test vector.

3. **Capture mode**: The circuit response is captured in the SFF. The SE signal is asserted and the captured response from SFF is shifted out.

The typical block diagram of a scan chain architecture is shown in Figure 2. All the SFFs are clocked with a scan clock, and the SO of one SFF is connected to SI of another SFF to form a scan chain.

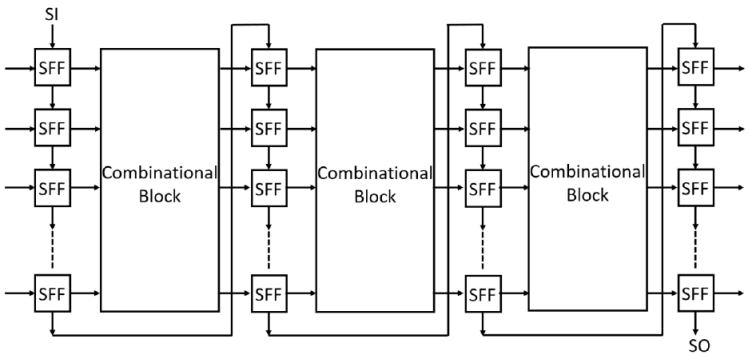

Figure 2. Scan Chain Architecture [12]

Figure 3 illustrates the Muxed-D Scan Approach assumed in this research. The three D flip flops are converted to SFFs by connecting a 2:1 Mux in the data input. X1, X2, X3 are PIs whereas Y1, Y2 are POs. Each DI of SFF is connected to a PPO of the combinational logic. The Q (output) port of each of the SFFs is connected to the combinational logic as a PPI. As explained above, the scan patterns are shifted into the SFF via SI by asserting the SE and putting the circuit in scan mode. Then, SE is de-asserted and the circuit is put into functional mode (normal operation). After the functional operation, SE is asserted again and the capture response of the DUT are shifted out from SFF from SO.

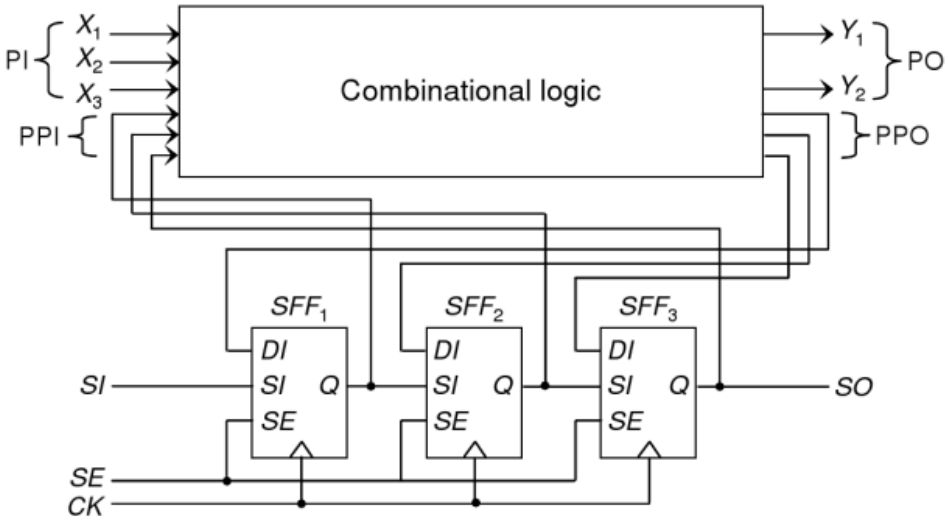

Figure 3. Muxed-D Scan Design [20]

1.4. At-speed Scan Clocking

There are two clocking schemes that are used generally to generate the test vector.

1.4.1. Launch on Shift (LOS)

This clocking scheme is also known as skewed load [20]. In this scheme, after the last shift clock pulse launches a transition, a capture pulse must immediately capture the output test response where the second capture clock pulse runs at functional speed(at-speed) . This requires SE signal to switch very fast between launch and capture clock pulse. Hence, SE signal has to be timed at functional frequency which is the speed of second capture clock pulse. Figure 4 shows the LOS scheme.

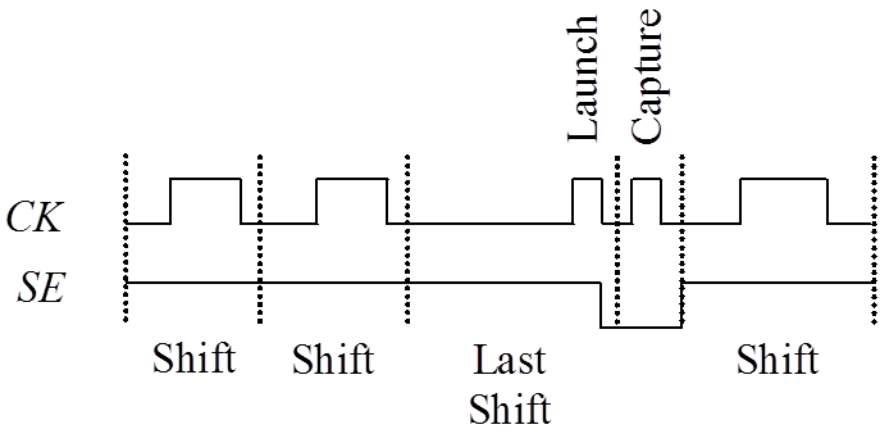

Figure 4. Launch on Shift Clocking Scheme [20]

1.4.2. Launch on Capture (LOC)

This clocking scheme is also known as broadside or double capture [20]. In this scheme, two at-speed launch and capture cycles are applied to launch a transition and capture the response. It doesn't have any speed related constraint on the SE signal as LOS testing scheme. After the test vectors are loaded into the SFFs, SE is de-asserted, and launch and capture cycles are applied after SE is stabilized. Because of the relaxed timing specification on SE signal, this scheme is mostly used even though it requires more test vectors resulting in lower fault coverage. Figure 5 shows the LOC scheme.

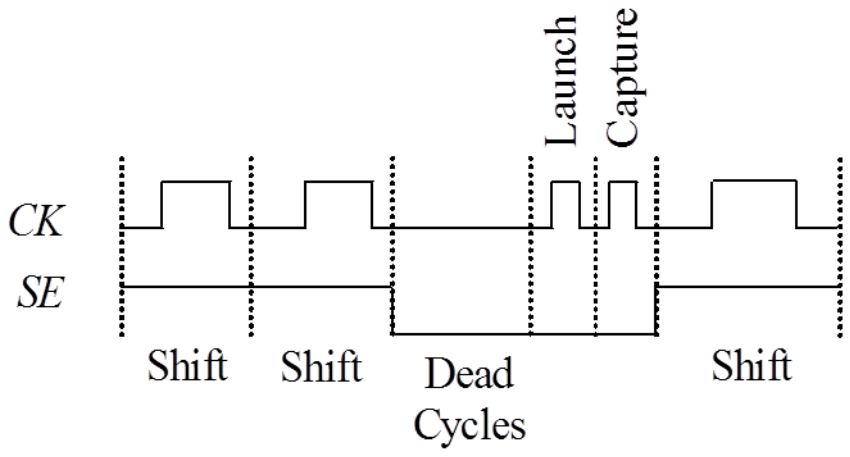

Figure 5. Launch on Capture Clocking Scheme [20]

1.5. Boolean Satisfiability

Boolean satisfiability (SAT) is used in a large scale for verification and testing of chips. A circuit is represented in Conjunctive Normal Form (CNF) [21]. SAT solvers are developed from Boolean Constraint Propagation and backtracking techniques [22]. Based on the conflict analysis learning, a test pattern is generated by the SAT solver.

A CNF is the logical AND of clauses where each clause is the ORs of literals. For example, for a two input OR gate with inputs as A and B and output with Z, corresponding CNF can be written as, (A+B+~Z) (~A+Z) (~B+Z). The logical OR operation is Z= A+ B. Here, A, B, Z are literals which are ORs and are ANDed with clauses. SAT solvers calculate the values of A, B and Z such that the CNF expression holds as 1. The different combinations with either(or both) A or B as 1 (A=0,B=1,Z=1 or A=1,B=0,Z=1 or A=1,B=1,Z=1) will generate the value of 1 to the CNF expression. Similarly, (~Z + A) (~Z + B) (~A + ~B + Z). Here, the CNF will generate 1 only when A=1, B=1, Z=1. Thus,

the main objective of SAT solver is to find out the values of different literals that gives the result of CNF expression to 1.

CNF clause with 2 variables is called 2CNF, and the clause with 3 variables is called 3CNF. 2CNF can be solved in polynomial time, whereas 3CNF is a NP-complete problem. Different heuristics are used to solve SAT-problems in reasonable time for any k-CNF.

In this research, SAT is used for the final justification and dynamic compaction of the KLPG algorithm [18]. For normal at-speed test, with two time frames for Launch and Capture, two variables (literals) are used by SAT for solving the necessary assignments assigned to each gate. Since we are using multiple at-speed test, more than two variables are required for the solution to solve in different time frames.

1.6. Structure of Thesis

An overall description of Pseudo Functional Test for K Longest Path per Gate (PKLPG) is given in section 2. We give a brief introduction of preamble, coda cycles along with the brief summary of KLPG algorithm. In section 3, we present the motivation behind the work. Section 4 describes the prior work that has been done in compaction of patterns generated by ATPGs. Implementation details are presented in section 5. Section 6 discusses the experimental results and Section 7 concludes the research with some future enhancements.

2. PSEUDO FUNCTIONAL TEST FOR K LONGEST PATH PER GATE

2.1. K Longest Path per Gate (KLPG)

The K Longest Path per Gate (KLPG) algorithm [1, 2] finds the K longest rising and falling path through each line in a combinational circuit. The search space for each target is the fan-in and fan-out of the target line as shown in Figure 6.

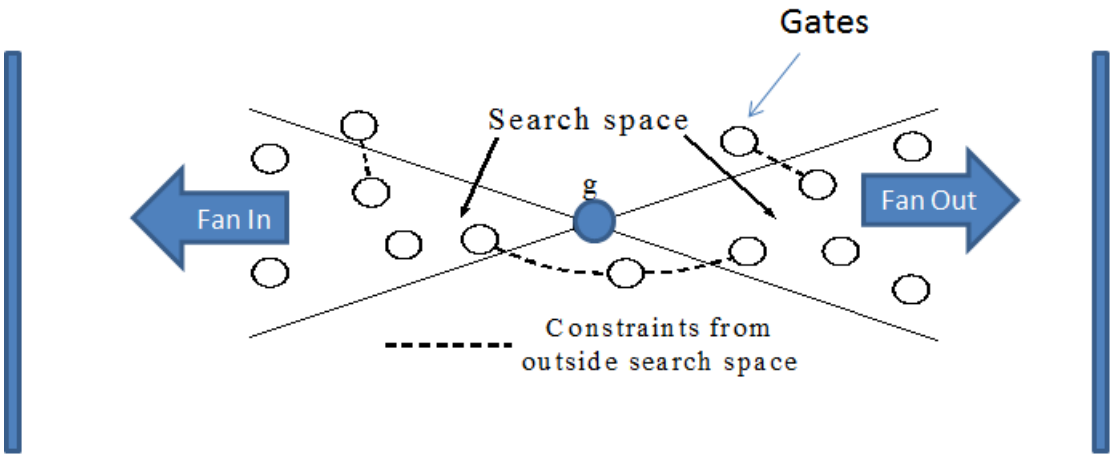

Figure 6. Search Space for a target g in KLPG [17]

A path starts from a launch point, Primary Input (PI) or a Pseudo Primary Input (PPI), and ends at a capture point, Primary Output (PO) or a Pseudo Primary Output (PPO) [20]. A PPI is a scan cell output and PPO is a scan cell input. KLPG algorithm has three major steps, path initialization, path growth and final justification. During initialization phase, fan-in and fan-out cones for each gate is determined along with the calculation of SCOAP metrics (controllability and observability) [25] for each gates. Controllability is

the measure of how easily a gate can be controlled from a PI or PPI, and observability is the measure of how easily a gate can be observed from a PO or PPO. The controllability of gates closer to PI or PPI are lower than the gates far from PI or PPI as it is easier to control the values of gates closer to the inputs. Similarly, the observability of gates closer to PO or PPO are lower compared to gates far from PO or PPO. The gates are levelized, on the basis of its maximum distance from PI/PPI.

The next step is path growth phase. The path is extended by adding one gate at a time leading to a formation of partial path. The partial path with the biggest Esperance (which means partial path is longest) is added to a partial path pool. A partial path becomes complete path when it reaches a PPO. After a complete path is found, which is the longest rising or falling path through a target gate, the path is justified to make sure that all the assigned values are compatible. Test patterns are generated during this phase. Figure 7 gives a brief overview of the KLPG algorithm.

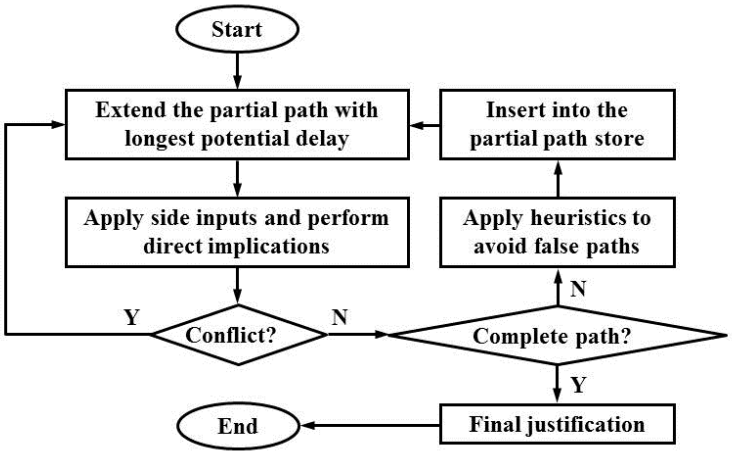

Figure 7. Overview of KLPG Path Generation [17]

The KLPG algorithm [17] can be described as:

1. Parse the input files and perform pre-processing steps such as computing SCOAP metrics on each gate.
2. **For** each target fault site, until K paths has been generated or no more are possible
3. Initialize the paths from the target fan-in cone Launch Points.
4. Add these to the Partial Path Store.
5. Extract the partial path with maximum *Esperance*.
6. Extend the extracted path.
7. Add side input constraints and perform Multi-Frame Direct Implications.
8. **If** a conflict is detected
9. Delete this partial path
10. **End If**
11. **Else If** Complete path formed
12. Perform Final Justification.
13. **End If**
14. **Else If** Complete Path is not formed
15. Apply false path elimination heuristics.
16. Update the *Esperance*.
17. Re-insert in a sorted fashion into the Partial Path Store
18. Go to step 6.
19. **End If**
20. **End For**

2.2. Pseudo Functional Test (PFT)

Pseudo Functional Test (PFT) is a structural delay test mechanism, in which traditional launch-on-capture delay testing is extended to additional launch and capture cycles. In this type of test, a scan (test) pattern is scanned into the circuit, and then multiple functional cycles are applied to it, where the last two cycles are at-speed launch and capture cycles. The cycles preceding at-speed cycles are called as preamble cycles.

After the test patterns are scanned in, the circuit switches from scan to functional mode. During this time, the currents in the off-chip connections are in the quiescent state.

The current rises abruptly when the at-speed cycles are applied. But, the off-chip inductance limits the speed that the current can be supplied, which introduces dI/dt power supply voltage droop in the chip [26]. Due to the droop, the chip operates slowly than in the functional mode and there is a chance that even good chips can fail the delay test. Figure 8 shows the introduction of voltage droop in delay test.

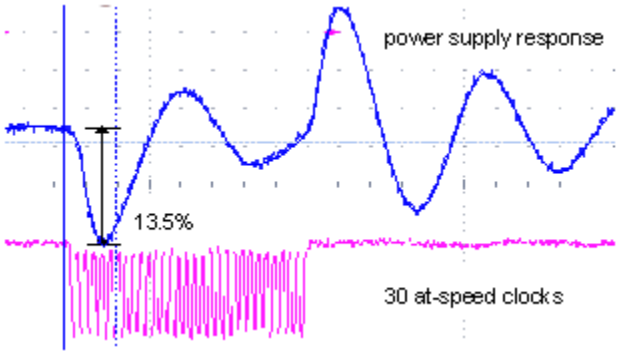

Figure 8. Drop in Power supply voltage during delay test [26]

Preamble cycles are scan or functional cycles that are slower than functional speed, but much faster than scan speed, to ramp inductor currents prior to the launch and capture of the delay test [17]. Due to the insertion of preamble cycles, circuit switching occurs over an extended period of time. It causes the off-chip power supply noise transient to die down prior to the at-speed launch and capture. This helps us in reaching an operating environment close to functional mode when the launch and capture take place. This allows the increase in delay test correlation between structural and functional models [17]. The test in KLPG using this cycles are called pseudo-functional KLPG test.

Preamble cycle is time frame expansion of traditional at-speed Launch-Capture test as shown in Figure 9 [17]. The transition is launched (Launch frame) and captured (Capture frame) in the last two time frames. The necessary assignments are justified to time frame 0, where the test vector is scanned in. The vector at each time frame is derived from the vectors at previous time frame. So, justification should be done till the first time frame (time frame 0) due to which as frames are added, the paths may decrease. The justification in preamble cycles is shown in Figure 10.

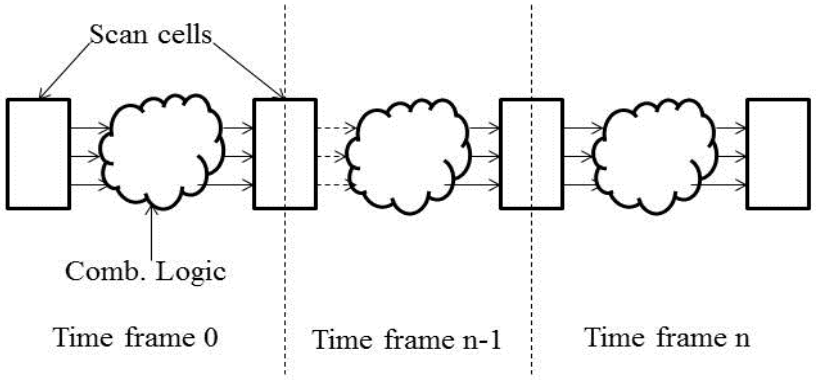

Figure 9. Time frame expansion of the circuit for PFT [17]

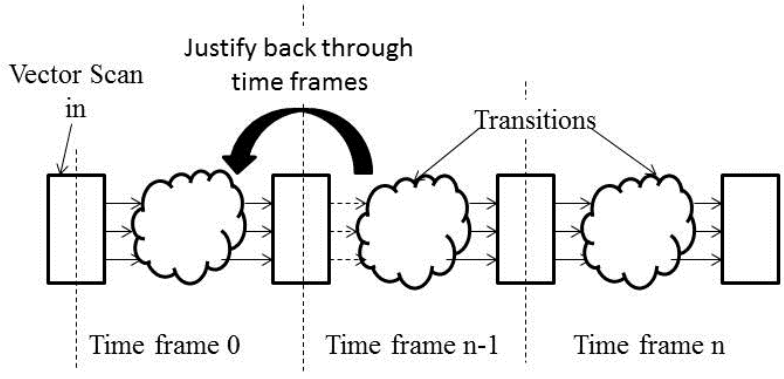

Figure 10. Extending the traditional LOC model for preamble cycles in PFT [17]

2.3. Coda Cycles

If the transition is captured in a non-scan cell, the response from a non-scan cell must be moved to a scan cell to read the captured value [17]. This extra capture cycle to propagate the captured value is called coda cycle. Coda cycle doesn't need to use the at-speed clock, and can be faster than scan clock. It can use a slow clock and is untimed. The ATPG uses Observability metric to select the most observable path to capture the transition (needing the fewest necessary assignments). If that fails, it tries the next-easiest, and so on. If the path is unobservable, the observability metric with assign infinite observability.

Figure 11 shows a condition when a transition is captured in a non-scan cell, and a low speed clock cycle is applied due to which the transition is capture in scan cell. The number of these low speed cycles vary depending upon when the value will be captured in the scan cell. In Figure 12, two or more coda cycles are applied until the value is captured and visible in the scan cell. In this research, we are performing Pseudo-functional test combined with coda cycles as shown in Figure 13. We are applying set of preamble cycles followed by at-speed cycles which performs the KLPG test, and then we are applying a set of coda cycles as needed to propagate a transition to scan cell. It can be clearly seen from the figure that at-speed cycles are very fast compared to the coda cycles and preamble cycles, where coda cycles is used for capturing the response and preamble is used for settling the power supply noise before the tradition at-speed launch-capture test.

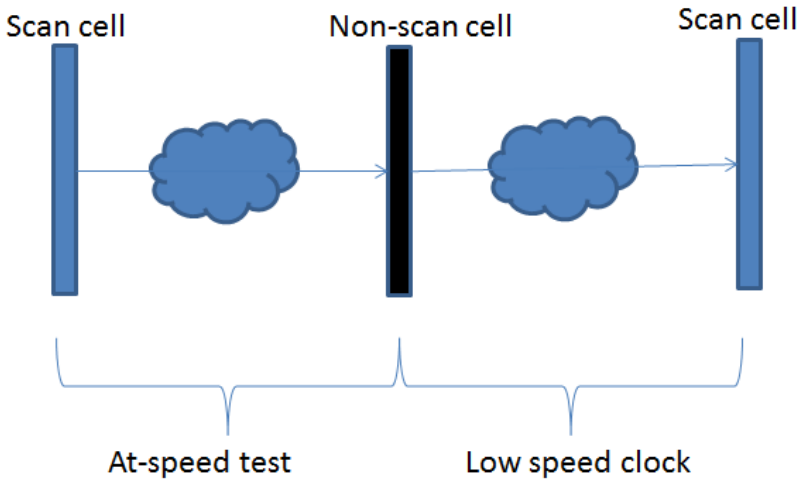

Figure 11. Propagation of Boolean value in coda cycle [17]

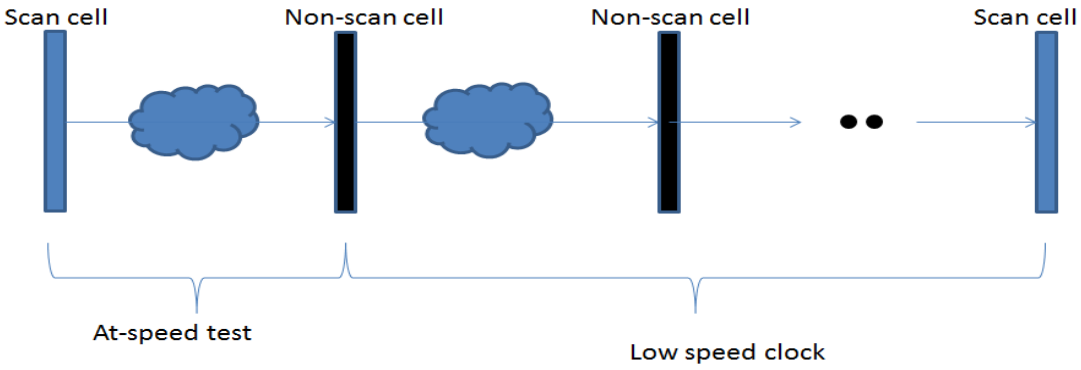

Figure 12. Multiple coda cycles [17]

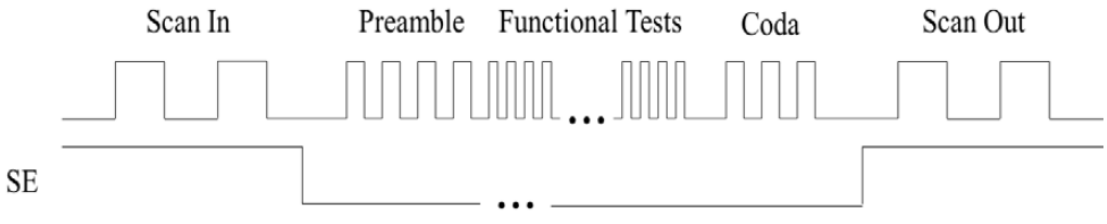

Figure 13. Coda cycle in PFT KLPG [23]

3. MOTIVATION

Scan design is mostly used these days in Design for Test for on-chip testing. The sequential elements in the chip is converted to SFFS and these SFFs are connected serially to form a scan chain. The test patterns should be loaded in the scan cells, and then the values are captured from the combinational circuit in normal mode, and again, the response is shifted out. The scan operation takes thousands of functional clock cycles (at-speed cycles). So, fewer the scan chains and test patterns, the faster will be the application test time along with saving the area overhead.

Some prior work has been done using multiple at-speed cycles. Multiple at-speed capture cycles was used to test the metastability of flip-flops in KLPG algorithm during delay test in [19]. K-longest paths per flip-flop test patterns were generated using three or more at-speed cycles, where longest path using Esperance on one cycle was feeding the longest path on next cycle. It was mainly targeted to test the time-borrowing between latches and flip-flop metastability rather than packing the tests in minimal number of patterns.

We are extending the research in KLPG algorithm [1, 2] using the observability metric for path delay testing [12]. After the longest-path is propagated from at-speed clock cycles for a target line using KLPG algorithm, the algorithm was modified accommodating observability metric. After the value was captured in a scan flip-flop, the path was propagated through the fan out of the captured scan flip-flop which had the minimum observability. This approach was mainly used when the capturing final flip flop

was not a scan flip-flop, and a definite number of coda cycles were used to propagate the response to a scan flip-flop. The path always grew through the branch with minimum observability and never ended at a non-scan flip-flop. We are using the same principle in this research. For a normal path, we are taking a predefined number of coda cycles so that after the at-speed path delay test is completed, the values are captured to the scan flip-flops. Multiple at-speed cycles are only used when there is conflict in the necessary assignments of the values of the gates and already present patterns.

The main motivation of the research is to pack many tests to one scan pattern as possible, due to the linear relationship of the number and size of scan patterns to the test time and tester cost. We try to compact as many paths using the at-speed cycles. Since, at-speed cycles are very fast compared to the scan cycles, we are trying to exploit if more paths can be placed in these multiple cycles after the scan-in operation. Based on the results, we can see that there are some paths which had conflicts with the patterns in the first time frame, but did not have conflicts in the different time frame. Due to this, the pattern count is reduced. As we unroll the circuit in multiple time frames, we observe that as the path count decreases, because when time frames are added, we need to sensitize the circuit to the previous time frames due to which some paths may get lost. But, we are mainly concerned with increasing the path to pattern ratio without losing fault coverage and trying to see if more paths can be compacted with reduced patterns and add at-speed cycles.

4. PRIOR WORK IN DYNAMIC COMPACTION

Significant work has been done on path delay testing. We are focusing our research on pattern generation related to compaction. An Automatic Test Pattern Generator generates boolean patterns for PIs and PPIs on Scan cells to detect the targeted faults. X are don't care values. Fewer the test patterns, smaller will be the testing time which is directly proportional to the test cost. If the test set size is more than the tester memory size, then the test patterns should be reloaded again until the desired fault coverage is obtained which makes testing even more expensive. Different techniques have been introduced to reduce the patterns and also decrease the test time by merging test patterns with non-conflicting values [4]. Compaction algorithm are generally classified as static or dynamic compaction. Static compaction is performed after test patterns are generated from the path finding procedure whereas dynamic compaction is done before the test patterns are generated from path finding procedure. The static and dynamic compaction techniques are described briefly in the following sections:

4.1. Static Compaction

In static compaction, if the same bit in test vectors are same or have a don't care value in at least one of them, then the vectors can be merged and compacted to form a new test vector [16]. In Figure 14, two vectors V1 and V2 are compacted to a new vector V3, as V1 and V2 are compatible with each other. 0 or X in same bit position is compacted to 0 and 1 or X in same bit position is compacted to 1.

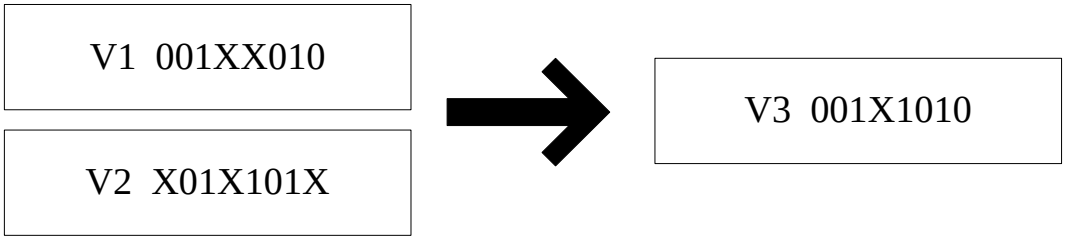

Figure 14. Static compaction of two 8-bit vectors [16]

Different techniques for reducing patterns without reducing fault coverage using static compaction was introduced in [5][7][8][9][10]. Greedy static compaction algorithm was used for KLPG test [1] as shown in Figure 15 where each pattern is compared with patterns in the compacted list, and is always merged with the first compatible pattern [11]. It was observed that the test size produced by greedy algorithm was nearly equal to the optimal results produced by a simulated annealing algorithm.

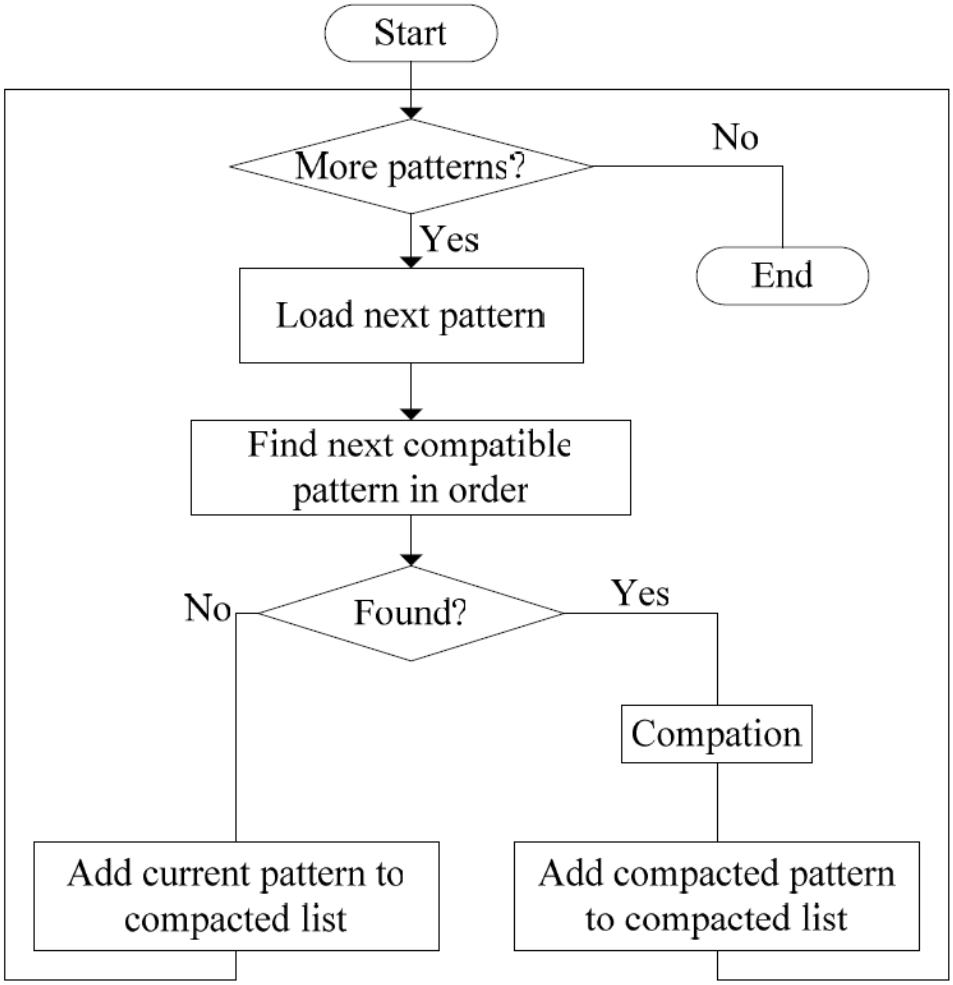

Figure 15. Greedy static compaction flow [16]

4.2. Dynamic Compaction

Dynamic compaction for KLPG algorithm was introduced in [3] which checks the compatibility between patterns, without reducing the fault coverage. The test vectors are only generated after the completion of dynamic compaction procedure. This approach was found to reduce pattern count (up to 4x compared to static compaction).

K-Longest Path Per Gate (KLPG) using Pseudo Functional Test was introduced in [6] were multiple preamble cycles were introduced prior to the traditional launch capture test, and dynamically compacted patterns were generated to test the longest paths into and out of each memory cells.

In this technique, paths are compacted based on their necessary assignments without fault simulation. The patterns are the set of necessary assignments for K longest rising and falling paths for a line. Figure 16 shows the example of a circuit with necessary assignment for targeting a STR fault on node S. The path propagation will be through the path which has maximum number of gates through it, which is a NOT, AND and OR gate. To observe a rising transition (0 to 1) along the longest path, we need a falling transition on NOT gate. Since the falling transition on NOT gate should be observed on AND_1 gate, the necessary assignments on the other input of AND_1 gate should be 11, which is the non-controlling value for an AND gate. Similarly, to observe this value through the OR gate, the second input of OR gate should be non-controlling (00). Since the output of AND_2 gate which is also the input of OR gate is 00, one of the inputs of AND_2 gate should be 00. So, B is assigned is 00, as S is the target gate with 0 to 1 transition. Hence, the different values assigned to all the gates in different time frames are the necessary assignments (patterns) for a target gate to find the K-longest path through it.

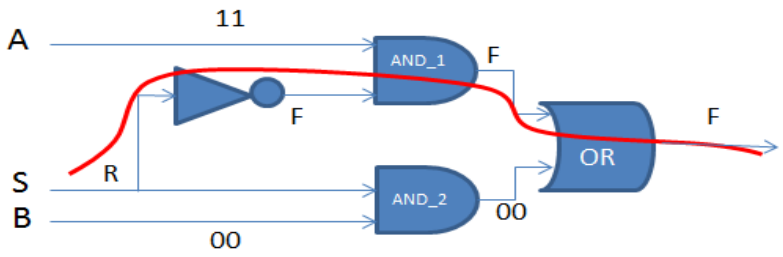

Figure 16. Necessary assignments for rising longest path in multiplexer [23]

After a path is generated, it is compared against the patterns in the pool where paths have been placed. If no conflict has been found, it combines the patterns to the first matched compatible pattern in the pool. If conflict is found with all the patterns in the pool, then a new pattern is created. The test vector generation using dynamic compaction is shown in Figures 17, 18 and 19. In figure 17, Path 1 is a complete path for falling transition through A with necessary assignments (in circles). Likewise, in figure 18, Path 2 is a complete path for rising transition through B, with necessary assignments as Xs. For both the paths, if necessary assignments are compatible with no conflict in value of assignments, then the necessary assignment can be combined into one, and a test vector can be generated after justification. The final test vectors are generated only after the completion of dynamic compaction procedure. This allows the flexibility for a pattern to be compacted in as much path as possible.

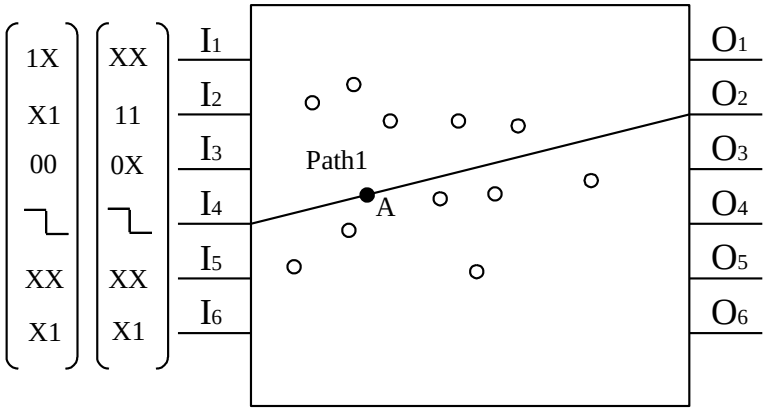

Figure 17. Vector pair and necessary assignments (circles) for Path1 [3]

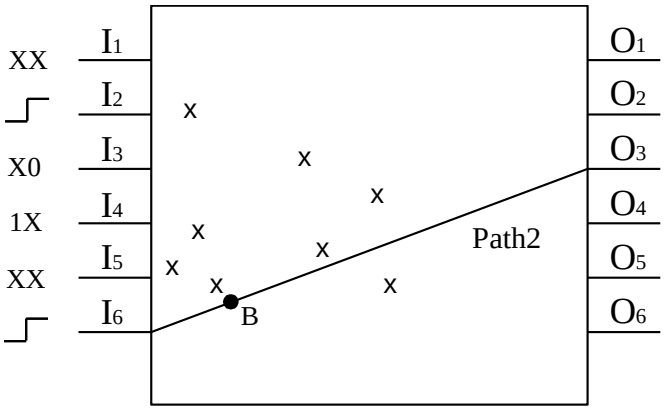

Figure 18. Vector pair and necessary assignments (Xs) for Path2 [3]

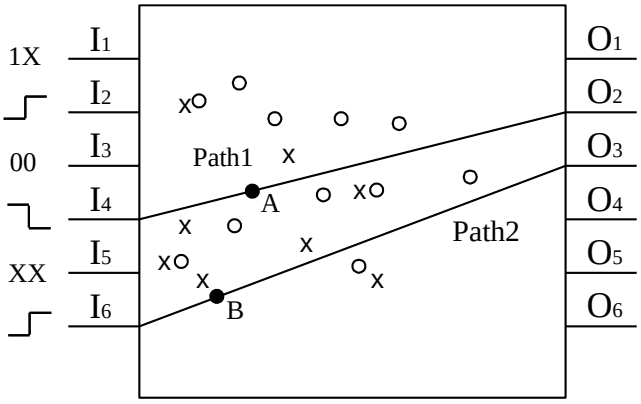

Figure 19. Vector pair and necessary assignments for Path1 and Path2 [3]

The flowchart for dynamic compaction is shown in figure 20. Dynamic Compaction was used in KLPG test with the pseudo code as mentioned below [3]:

Procedure *klpg_dc*( )

1. Initialize pattern pool *POOL* as empty.
2. Use KLPG to generate a successful longest path *I* through a line, resulting in pattern *F*. F contains all necessary assignment information before justification. Justification of *F* is performed to check that the path is sensitizable, but the resulting primary input values are not stored. If no more paths can be generated or we have enough paths, go to step 4. Otherwise go to step 3.
3. Call procedure *dyn_compact*(*F*,*POOL*). Go to step 2.
4. Do final justification for all patterns in *POOL* one by one to generate the final vectors. Procedure is finished.

Procedure dyn_compact(F, POOL)

1. If POOL is empty, insert pattern F into POOL and return. Otherwise set pointer P to the first pattern in POOL and go to step 3.
2. Set pointer P to the next pattern in POOL. If P is pointing to empty (the end of POOL), go to step 6. Otherwise go to step 3.
3. Do conflict check between F and P. If there is a conflict, go to step 2. Otherwise go to step 4. Combine two sets of necessary assignments F and P, and save them as K. Check for direct implication conflicts in K. If no conflicts, go to step 5. Otherwise delete K and go to step 2.

5. Do final justification for K. If K passes final justification, update P by combining necessary assignments of F into it and return. Otherwise delete K and go to step 2.
6. Insert F into POOL as a pattern. Return.

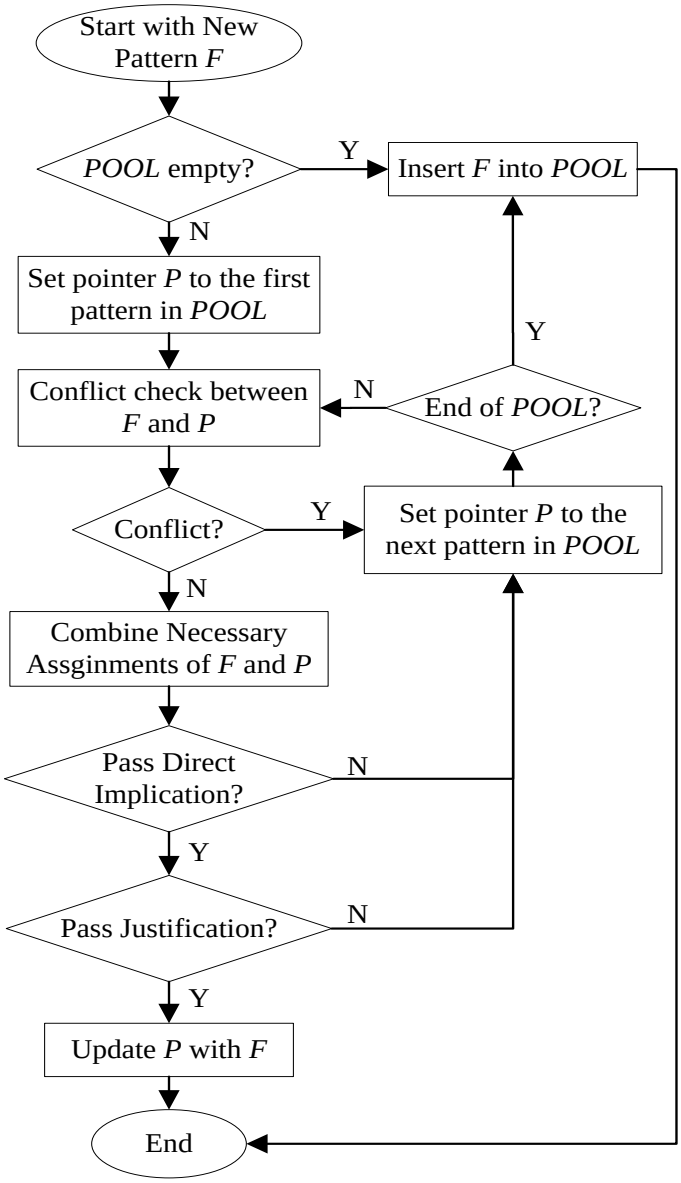

Figure 20. Flowchart of dynamic compaction algorithm [16]

5. IMPLEMENTATION

5.1. Key Decision

The KLPG algorithm using observability metric [12] has been extended in this research. When a path is generated, we are identifying a coda path and keeping necessary assignments of coda path in path pool. We are not keeping the necessary assignments of preamble. Since preamble and coda are trying to pick the easiest path, we are making an assumption that timing of preamble, at-speed and coda cycles are synchronized. We don't have a path for preamble cycle as the number of preamble depends on the presence of conflict with the patterns in the pool.

5.2. At-Speed Path Generation

For the implementation of multi-cycle at-speed test algorithm for dynamically compacting patterns, changes were made to the existing KLPG algorithm in Codgen [12] where the total number of frames were fixed. Initially, the total time frames was entered by the user [1][2]. It the total number of FRAMES was 5, then at-speed path was always generated in the last two time frames using Esperance metric. The last two cycles are Launch and Capture, and the first three cycles are preamble cycles. When observability metric was added in [12], then the total number of coda cycles for each path was unique and was entered in the system. Number of preamble and coda cycles were fixed in the system.

The existing algorithm was modified in such a way that the at-speed along with preamble and coda cycles path generation are adjusted automatically. In the implemented system, for normal case, preamble is taken as zero, and we try to compact paths in the first two time frames using at-speed Esperance metric. Depending upon the number of time frames entered, after the response is captured, the remaining cycles are slow speed capture cycles, which propagates the captured value in Scan Flip Flop. If the KLPG test is able to compact in the first time frame itself, then the normal initial condition with definite coda cycles are used for the path propagation. If the patterns are not compatible, then the self-adjustment of preamble, at-speed and coda cycles comes into picture. If the test is not able to compact in the first time frame, the test is re-generated in the different time frame, which modifies the initial flow of the algorithm in Codgen.

The existing algorithm [3] generated a new test pattern if the pattern was not compatible with the patterns in the pattern pool. The modified algorithm in this research will regenerate a path multiple times through the same target but in a different time frame unless the limit of searching in time frames have reached. So, this technique is majorly targeting the generation of few scan patterns.

5.3. Path Propagation Logistics

For a given target fault, we want to compact paths in as few scan patterns as possible, as the number of scan patterns is proportional to the testing time and cost. We want to make use of multiple at-speed cycles in such a way that if a target fault can’t be detected in first time frame, then it could be detected in other consecutive time frames.

We are doing time frame expansion of the circuit. Let us consider the example circuit as shown in Figure 21. In this circuit, we are trying to find a complete path for a rising transition in Scan Flip Flop A. We are assuming that we are unrolling the circuit three times. For normal case, the first two time frames represent the at-speed Launch and Capture frames whereas the third and fourth time frames are coda cycles where the captured values are stored after two at-speed cycles. The circuit for normal operation behaves as shown in Figure 22. During the at-speed Launch-Capture time frame, the longest path through the target gate is generated. After the target value is captured in the flop, for the remaining two time frames, the path will follow the most observable (easiest) path which will generally be the path with minimum number of gates through it. So, the path covered would be, A-b-c-A-d-A-d-A as shown in Figure 22.

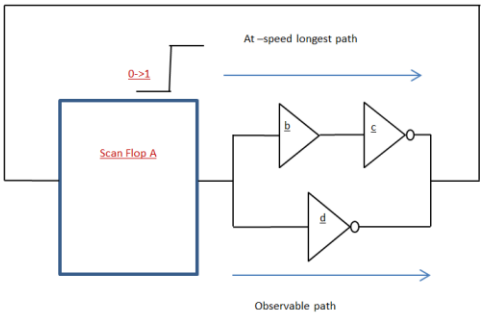

Figure 21. Example Circuit

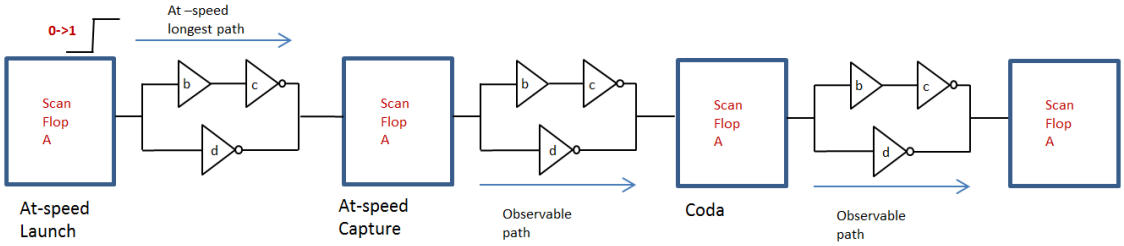

Figure 22. Path propagation with one capture and two coda cycles

After finding the longest path through the target gate A with two coda cycles, we have a set of necessary assignments through that path which are the values assigned to each gates while propagating that path. If it is the first path of the KLPG algorithm, then the patterns of this path is stored in the pattern pool as the first pattern. But, if it is not the first path, then the patterns for this path are compared with the patterns in pattern pool. If there isn't any conflict with the patterns in pool, then it combines to the first compatible pattern found in the pool. But, if conflict is found, then the path is re-generated for the same target fault in another time frame. In this problem for the circuit shown in Figure 21, the path is propagated again with one preamble, one at-speed launch and capture and one coda cycle as shown in Figure 23. As shown in the figure, the path is propagated through A-b-c-d-A, where the longest at-speed time frame is calculated in the second and third time frame, and the captured value is propagated to scan flip-flop using one coda cycle.

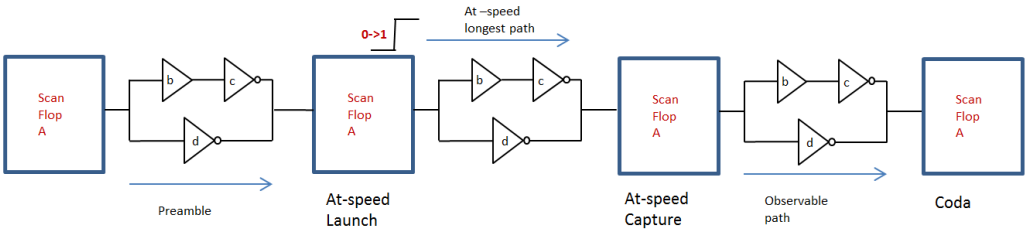

Figure 23. Path propagation with one preamble, one capture and one coda cycle

Since we are unrolling the circuit three times, we will continue the process until coda cycle is zero if conflict is found. After finding path with one preamble and two coda, we then compare the necessary assignments of that path with the patterns in pool. If there is no conflict in this time frame, then, we combine the pattern (which is the pattern for the second time frame) with the most compatible pattern in the pool. But, if there is conflict with patterns in pool even for this case, then we propagate path again through the same target gate with two preamble and zero coda cycles. The implementation is shown in Figure 24. In this case, the path propagated will be A-b-c-A which is the longest path through target fault A with no coda cycle and two preamble cycles.

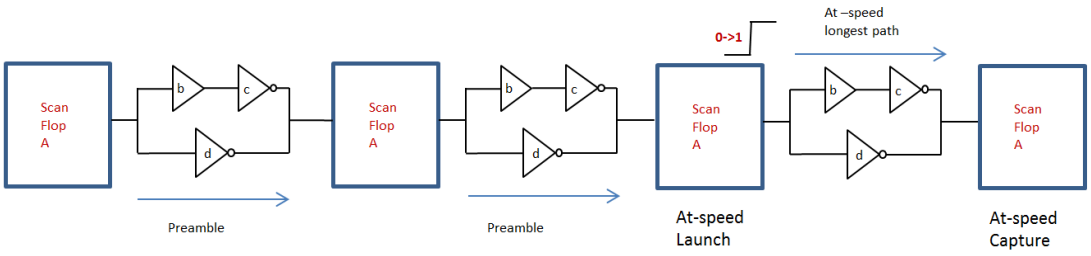

Figure 24. Path propagation with two preamble and one capture cycle

Now, for the path generated in third time frame, if conflict is not found, then the patterns are combined with the first compatible pattern. But, if there is still conflict, then we will take the pattern for the normal case for that path, with first at-speed test and two coda cycles. We save the patterns for the normal condition so that in the presence of conflicts in the paths generated for different time frames, we will place the initial saved pattern in the pool. This will help save time for the path propagation, and will also cover the path in the pool which preserves the coverage of the system. When the saved pattern is placed in the pool, a new pattern in the pattern pool is created.

These three scenarios can be represented in the form of timing diagrams as shown in Figure 25, where the path is propagated at different at-speed cycles.

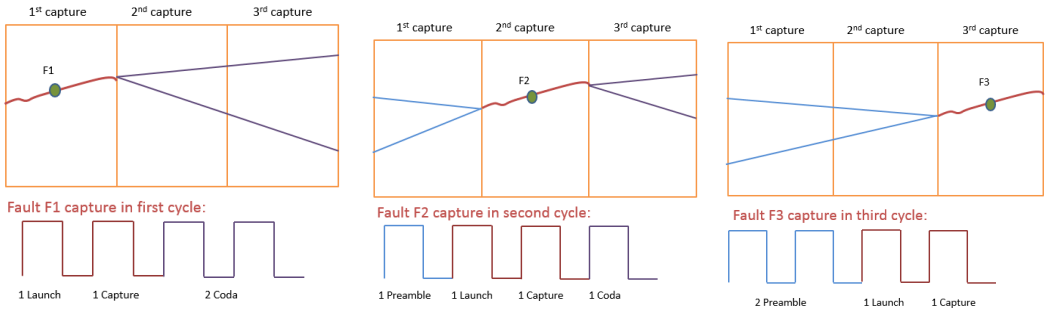

Figure 25. Path propagation in first, second and third at- speed cycles

5.4. Proposed Algorithm

The following section explains the pseudocode and Figure 26 presents flowchart of the proposed algorithm.

Algorithm:

1. Enter total time frames to be expanded as FRAMES.

2. For a new path, total coda cycles, c = FRAMES – 2

3. Is c > = 0?

4. Yes: Propagate with 1 L/C(1 Launch and Capture) and c coda cycles.

5. Save Path with 1 L/C and c coda cycles. Go to step 7.

6. No: Goto Step 9.

7. Check if pattern pool is empty.

8. Yes: Load the pattern with 1 L/ C and c coda from step 5.

9. Place the pattern with 1 L/C and c coda to pattern pool.

10. No: Go to step 11.

11. Check for conflict between necessary assignments of this path and patterns in pattern pool.

12. Yes: L/C = L/C +1, Preamble= Preamble + 1; Coda = Coda – 1. Go to step 3.

13. No: Combine necessary assignments of this pattern with compatible pattern in the pool. Go to step 2 for a new path with new target.

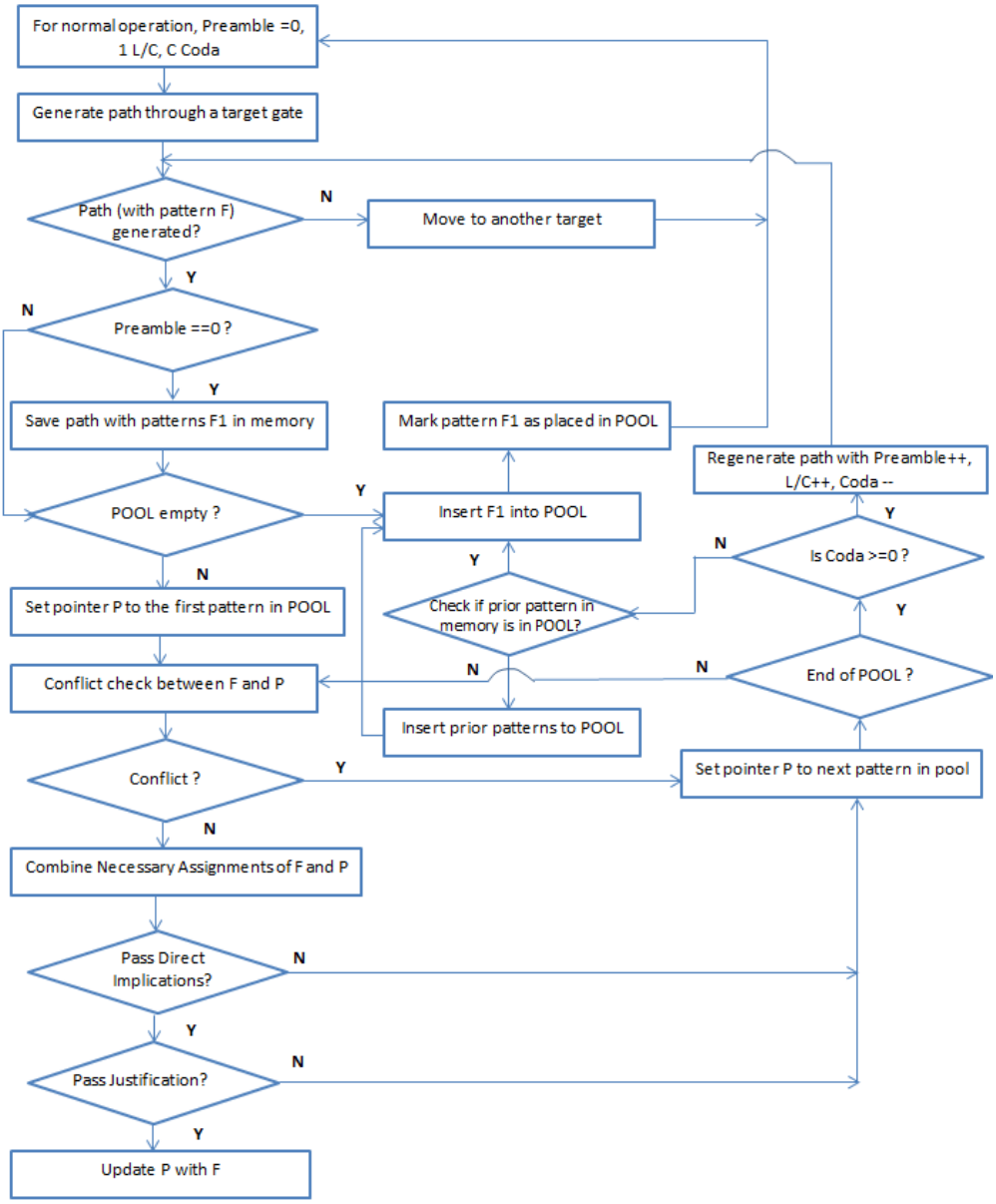

Figure 26. Flowchart of Proposed Compaction algorithm

5.5. Code Flow of Implemented System

Some snapshot for the path generation implementation for the circuit is shown in Figure 27.

FRAMES= 4 (2 coda cycles in normal case)

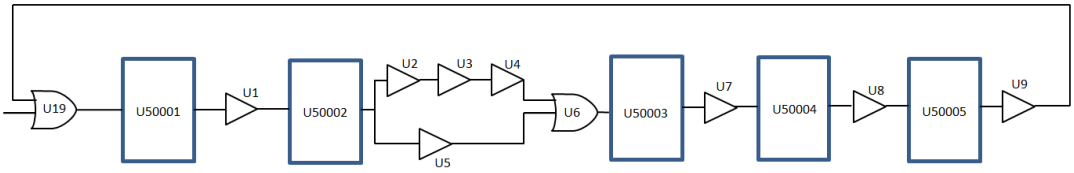

Figure 27. Example circuit used in implementation

Gates :

U 50004 G: (3) (R) -> U 8 G: (13) (R) -> U 50005 G: (21) (R) -> U 9 G: (14) (R) -> U 19 G: (15) (R) -> U 50001 G: (17) (R) -> U 1 G: (6) (R) -> U 50002 G: (18) (R)

CONFLICT found for patterns in pool for time frame : 0

Path Regeneration with time frame: 1 with Gate U 50004 with Rising Transition

time frame: 1

Path :

U 50004 G: (3) (R) -> U 8 G: (13) (R) -> U 50005 G: (21) (R) -> U 9 G: (14) (R) -> U 19 G: (15) (R) -> U 50001 G: (17) (R)

The path should be RISING

Gates :

U 50004 G: (3) (R) -> U 8 G: (13) (R) -> U 50005 G: (21) (R) -> U 9 G: (14) (R) -> U 19 G: (15) (R) -> U 50001 G: (17) (R)

CONFLICT found for patterns in pool for time frame : 1

Path Regeneration with time frame: 2 with Gate U 50004 with Rising Transition

time frame: 2

Path :

U 50004 G: (3) (R) -> U 8 G: (13) (R) -> U 50005 G: (21) (R)

The path should be RISING

Gates :

U 50004 G: (3) (R) -> U 8 G: (13) (R) -> U 50005 G: (21) (R)

NO Conflict with time frame: 2

We are taking total time frames as 4 in this example, which means that for normal operation, the first two time frames are at-speed launch and capture, and the last two frames are slow speed coda cycles. As seen above, after one complete path is propagated, we have a set of necessary assignments for that path. The pattern pool is scanned to check if the patterns of the path is compatible with any path in the pattern pool. If it is not compatible, then the path is regenerated again, in a different time frame for the same target gate.

In a different time frame, after path generation, the patterns in the pattern pool are scanned again to see if it matches with the necessary assignments of the path. In this example, the patterns are not compatible for the second time which was generated with

one preamble and one coda cycle. Then, the path is generated again with two preambles and zero coda. This time the patterns of the path is compatible with patterns in the pool.

The path can be regenerated until the number of coda cycles is zero or greater than zero. If path was not matched even with zero coda cycles, then the patterns with the initial case taking one preamble and two coda cycles were placed in the pattern pool.

6. EXPERIMENTAL RESULTS

The modified dynamic compaction algorithm for KLPG test has been implemented in C++ on an Intel Core i7 machine. Experiments have been carried on ISCAS89 benchmark circuits. Robust test has been considered with different values of FRAMES that determine how many times the circuit is unrolled.

Let's consider a path that is generated in first time frame, but it has conflict with patterns in the pattern pool. Then, the path is regenerated again through the same target line in other time frames. But, if the path can't be generated (sensitized) in other time frames, then we place the path that was originally there in first time frame (we are saving) in the pattern pool. Now, each of those paths is placed as a new pattern. These paths fall under column as "Paths as new patterns due to no paths in other time frames".

There can also be path where conflict is found in first time frame. So, it is regenerated in other time frames. But, if conflict is found in the path through the same target gate in other time frames too, then we place the paths in the first time frame (which we saved) to the pattern pool as new pattern. These paths fall under column as "Paths as new patterns due to conflict in other time frames".

In the dynamic compaction procedure, we check if there is any conflict between the necessary assignments of a path with respect to patterns in the pool. If there is no conflict, then we combine the NAs of the path with the most compatible pattern in the pool. A situation was found, when after the NAs of the path was combined with the best compatible pattern, and when the combined pattern was justified by SAT, it was failed.

The pattern was then compared with remaining patterns in the pool. It was either not able to find compatible pattern, or even when it found non-conflicting pattern and the two patterns were combined; the justification of that pattern was failed by SAT. So, in that case, the pattern for that path is placed as a new pattern in the pattern pool which is under the column as "Paths as new patterns because not justified with SAT". The pattern count with this condition is indeed difficult to meet and was found mainly for circuits with FRAMES=3, and was rare with FRAMES=4 and FRAMES =5. Table 1 shows the total paths and pattern count for different circuits for FRAMES=3.

Table 1. Total paths and patterns for FRAMES =3, for Robust case

	FRAMES=3										
	#Path						Modified(with at-speed compaction)			Without at-speed compaction	
	#Path in 1st TF	#Path in 2nd TF	#Path as new pattern due to no path in other TF	#Path as new pattern due to conflict in other TF	#Path as new pattern because not justifiable by SAT	#Total Path	#Pattern	Path: Pattern	FC %	#Pattern	Path: Pattern
s1488	93	1	6	8	0	108	15	7.2	9	16	6.75
s1494	83	1	9	5	1	99	16	6.19	16.01	17	5.82
s1423	86	3	3	1	2	95	7	13.57	13.8	10	9.5
s5378	236	30	6	0	0	272	7	38.86	14.58	37	7.35
s9234	173	0	4	0	2	179	7	25.57	3.23	7	25.57
s13207	304	0	6	2	1	313	10	31.3	11.34	10	31.3
s15850	480	13	6	2	11	512	20	25.6	8.92	33	15.52
s38417	876	3	8	1	2	890	12	74.17	2.8	15	59.33
s38584	1858	18	12	5	4	1897	22	86.23	2	40	47.43

In Table 1, the paths in 1st TF are the paths in first time frame which has zero preamble, one at-speed launch and one at-speed capture with one coda cycle. Similarly, the paths in 2nd TF are the paths in second time frame, which has one preamble, one at-speed Launch and one at-speed Capture and zero coda cycles respectively. If we consider circuit s5378 from Table 1, out of 272 total paths in the pattern pool, 236 paths are placed in first time frame and 30 paths are placed in the second time frame. These 30 paths had conflict in first time frame, and were regenerated in second time frame. Later, they didn't have conflict with one preamble and zero coda cycle. So, these paths in second time frame were compatible with the paths in first time frame after re-doing the KLPG test, and were placed together with the most compatible pattern. There were also other 6 paths which were generated in first time frame but had conflict with patterns in the pattern pool. But, these paths couldn't be sensitized in second time frame through the same target line. So, they were placed in the first time frame itself, where each of those 6 paths were placed as new patterns.

The total number of patterns found was the sum of patterns which constitute of placing the first path in the pattern pool as the first pattern, total patterns created because the paths were not sensitized in different time frames, the new patterns formed due to conflicts of the path with the coda time frames path and patterns resulting from not being justified by SAT. As can be seen from Table 1, for s5378 benchmark circuit, the total patterns are 7, which is the sum of 1 and 6 patterns. The first pattern is the pattern resulting from placing all the paths from the first time frame and second time frame which didn’t

have conflict with each other. Similarly, the total number of paths was 272, which is the sum of 236, 30 and 6 paths.

If multi-cycle at-speed approach was not used in this technique, the last two columns in Table 1 represent the total patterns and paths per pattern for each benchmark circuits. In that case, if the patterns had conflict with the patterns in the pattern pool, then, instead of regenerating a path through the same target in different time frames, that pattern would be created as a new pattern. For example, s1488 would have 16 patterns (15 patterns using this approach saving 1 pattern), s1494 would have 17 patterns (16 patterns using this approach saving 1 pattern), s1423 would have 10 patterns (7 patterns using this approach saving 3 patterns), s5378 would have 37 patterns (7 patterns using this approach saving 20 patterns), s15850 would have 33 patterns (20 patterns using this approach saving 13 patterns), s38417 would have 15 patterns (12 patterns using this approach saving 3 patterns) and s38584 would have 40 patterns (22 patterns using this approach saving 18 patterns). Hence, it can clearly be seen that this approach produces fewer patterns with increased paths per pattern compared to the compaction technique if at-speed compaction was not done.

When we place the first path in the pattern pool, that path has one at-speed Launch and Capture and one slow-speed coda cycle. When we perform KLPG tests for other target lines, we scan the pool for time frame one and if all patterns conflict with the new set of NAs (path), then when we generate the path in the second time frame and attempt to compact into the second time frame. The path in the second time frame was found to have conflicts with the already placed patterns in the pattern pool which had coda cycles. It was

also found that the first placed pattern in the pool was able to accommodate maximum number of paths. Figure 28 presents the snapshot of the paths compacted in pattern pool for s5378. Figure 29 shows the log file output showing the total number of paths for s5378 under different conditions described by Table 1.

```
Paths compacted in pathpool 13FD4068 :  133
Paths compacted in pathpool 1524C2F0 :  64
Paths compacted in pathpool 153BDEF0 :  27
Paths compacted in pathpool 1588FA30 :  14
Paths compacted in pathpool 15758BE0 :  11
Paths compacted in pathpool 15A36950 :  21
Paths compacted in pathpool 15F95460 :  2
Number of patterns in pathpool :7
Total paths: 272
```

Figure 28. Total patterns in the pattern pool for s5378, FRAMES=3

```
Paths in 1st TF : 236
Paths in 2nd TF : 30
Paths as new patterns due to no paths in other TF : 6
Paths as new patterns due to conflict in other TF : 0
Paths as new patterns because not justified by SAT : 0
```

Figure 29. Paths recorded for s5378, FRAMES=3

The total number of patterns for KLPG test for s5378 is 7 based on Figure 28 and Table 1, covering a total of 272 paths. Some pattern covers 133 paths, whereas some patterns just cover 2 paths. Here, we can see that the first pattern which was created at first when there were no initial patterns in the pattern pool, was able to accommodate larger number of paths. It could be the case that there are some patterns which is able to cover

almost majority of the faults, and hence that pattern is able to compact larger number of paths. It was also found for few circuits, patterns that are created when there are no paths in other time frames are able to cover all the remaining paths in the circuit.

Table 2 shows the total paths and pattern count for different circuits for FRAMES=4. For this configuration, the paths in 1st TF are the paths in first time frame with zero preamble, one at-speed Launch and one at-speed Capture with two coda cycles. The paths in 2nd TF are the paths in second time frame, which has one preamble, one at-speed Launch and one at-speed Capture and one coda cycle. Similarly, the paths in 3rd TF are the paths in third time frame with two preamble, one at-speed launch and zero coda cycle. Figure 30 presents the snapshot of the paths compacted in pattern pool for s5378 and Figure 31 shows the log file output showing the total number of paths for s5378 under different conditions described by Table 2 respectively (for FRAMES=4).

Table 2. Total paths and patterns for FRAMES =4, for Robust case

	FRAMES=4											
	#Path							Modified(with at-speed compaction)			Without at-speed compaction	
	#Path in 1st TF	#Path in 2nd TF	#Path in 3rd TF	#Path as new pattern due to no path in other TF	#Path as new pattern due to conflict in other TF	#Path as new pattern because not justifiable by SAT	#Total Path	#Pattern	Path: Pattern	FC %	#Pattern	Path: Pattern
s1488	33	5	1	3	0	1	43	5	8.6	10.38	11	3.91
s1494	34	1	0	3	0	0	38	4	9.5	11.1	5	7.6
s1423	22	0	0	1	0	1	24	3	8	7	3	8
s5378	228	3	0	2	0	0	233	3	77.67	3.82	6	38.83
s9234	126	0	0	3	0	0	129	4	32.25	3	4	32.25
s13207	85	0	0	1	0	0	86	2	43	6	2	43
s15850	215	4	2	9	2	5	237	17	13.94	5.75	23	10.3
s38417	584	0	0	3	1	0	588	5	117.6	2	5	117.6
s38584	1069	5	3	9	1	3	1090	14	77.86	2.42	22	49.55

```
Paths compacted in pathpool 147C8C98 :  188
Paths compacted in pathpool 158B01D8 :  41
Paths compacted in pathpool 16FD2CF8 :  4
Number of patterns in pathpool :3
Total paths: 233
```

Figure 30. Total patterns in the pattern pool for s5378, FRAMES=4

```
Paths in 1st TF : 228
Paths in 2nd TF : 3
Paths in 3rd TF : 0
Paths as new patterns due to no paths in other TF : 2
Paths as new patterns due to conflict in other TF : 0
Paths as new patterns because not justified by SAT : 0
```

Figure 31. Paths recorded for s5378, FRAMES=4

From Figure 30, for s5378, we could say that most of the patterns are compacted in the first pattern itself, accommodating a total of 188 patterns, and remaining of the paths are compacted in two other patterns that were generated from two paths that had conflicts in preceding time frames and couldn't sensitize a path for the same target in following time frames.

As observed from Table 2, considering circuit s38584, the total patterns are 14, which is the sum of 1, 9,1 and 3 patterns. The first pattern refers to the paths that were placed in first, second or third time frame which didn't have conflict with each other. Likewise, the total paths in the circuit are 1090, which is the sum of 1069, 5, 3, 9,1 and 3 paths.

Using multi-cycle at-speed compaction technique, the patterns count is smaller compared to the technique if at-speed compaction was not used. The behavior was similar as observed using FRAMES = 3. From Table 2, we can see that, if this technique was not used, s1488 would have 11 patterns (5 patterns using this approach saving 6 patterns), s1494 would have 5 patterns (4 patterns using this approach saving 1 pattern), s5378 would have 6 patterns (3 patterns using this approach saving 3 patterns), s15850 would have 23 patterns (17 patterns using this approach saving 6 patterns) and s38584 would have 22 patterns (14 patterns using this approach saving 8 patterns).

Table 3 presents the total paths and pattern count for different circuits for FRAMES=5. In this configuration, the paths in 1st TF are the paths in first time frame with zero preamble, one at-speed Launch and one at-speed Capture and three coda cycles. The paths in 2nd TF are the paths in 2nd TF with one preamble, one at-speed Launch and one

at-speed Capture and two coda cycles. Likewise, the paths in 3rd TF are the paths in third time frame with two preamble, one at-speed Launch and one at-speed Capture and one coda cycle. And, the paths in 4th TF are the paths in fourth time frame, with three preamble, one at-speed launch and one at-speed capture with zero coda cycle. Figure 32 presents the snapshot of the paths compacted in pattern pool for s5378 and Figure 33 shows the log file output showing the total number of paths for s5378 under different conditions described by Table 3 (for FRAMES=5) respectively.

Table 3. Total paths and patterns for FRAMES =5, for Robust case

	FRAMES=5												
	#Path								Modified(with at-speed compaction)			Without at-speed compaction	
	#Path in 1st TF	#Path in 2nd TF	#Path in 3rd TF	#Path in 4th TF	#Path as new pattern due to no path in other TF	#Path as new pattern due to conflict in other TF	#Path as new pattern because not justifiable by SAT	#Total Path	#Pattern	Path: Pattern	FC %	#Pattern	Path: Pattern
s1488	41	0	0	0	5	0	0	46	6	7.67	7	6	7.67
s1494	14	0	2	0	1	0	0	17	2	8.5	7	4	4.25
s1423	30	0	0	0	1	0	0	31	2	15.5	7	2	15.5
s5378	162	0	2	4	2	0	0	170	3	56.67	4	9	18.89
s9234	52	0	0	0	1	0	0	53	2	26.5	2	2	26.5
s13207	24	0	0	0	1	0	0	25	2	12.5	2	2	12.5
s15850	106	0	0	0	6	0	0	112	7	16	5	7	16
s38417	175	0	0	0	2	0	0	177	3	59	2	3	59
s38584	560	3	0	0	7	0	1	571	9	63.44	2.11	12	47.58

```
Paths compacted in pathpool 0B1E2F10 :  139
Paths compacted in pathpool 0CAD7DA8 :  28
Paths compacted in pathpool 0E3CDC60 :  3
Number of patterns in pathpool :3
Total paths: 170
```

Figure 32. Total patterns in the pattern pool for s5378, FRAMES=5

```
Paths in  1st TF : 162
Paths in  2nd TF : 0
Paths in  3rd TF : 2
Paths in  4th TF : 4
Paths as new patterns due to no paths in other TF : 2
Paths as new patterns due to conflict in other TF : 0
Paths as new patterns because not justified by SAT : 0
```

Figure 33. Paths recorded for s5378, FRAMES=5

As seen in Figure 32, there were some paths which had conflicts in first time frame but didn't have conflicts in third and fourth time frames. This could lead to increase in path to pattern ratio. Larger number of paths were compacted in the first created pattern itself as shown in Figure 33, which shows the similar results compared to FRAMES=3 and FRAMES=4.

For s38417 circuit observed from Table 3, the total patterns generated are 3, which is the sum of 1 and 2 patterns. The total paths are 177 formed from 175 and 2 paths.

The patterns are saved for this configuration for FRAMES =5 using this technique of multi-cycle at–speed compaction. If this technique was not used, s1494 would have 4 patterns (2 patterns using this approach saving 2 patterns), s5378 would have 9 patterns (3

patterns using this approach saving 6 patterns) and s38584 would have 12 patterns (9 patterns using this approach saving 3 patterns).

We can see that as the number of gates in a circuit increases, the number of paths and patterns increase proportionally. The larger the paths, the larger are the patterns and vice versa. When the circuit is unrolled in multiple time frames, the paths decreases reducing the number of patterns due to the dependence on preamble cycles for path sensitization and justification. This is observed in Figure 34 and 35. The path to pattern ratio was found to increase after implementation of multi-cycle at speed test for different configuration of FRAMES= 3, 4, 5.

Fewer paths were found to be compacted in time frames following the first time frame in all the three different configurations. This is basically due to three different scenarios:

1) After a path is generated, and if it has conflicts with paths in the pattern pool, then the path is regenerated again through the same target line in a different time frame. But, there can be a case that the path can't be generated or sensitized in different time frames. This could be due to the added constraint of preamble cycle to justify till the initialization vector, or due to the constraints of fixed Primary Inputs. Due to this, the paths that were generated in first time frame that were saved initially are placed in the pattern pool.

2) When a conflict has been found for patterns after a path is generated, the path is regenerated again in a different time frame. But, there could be a case that the path in a different time frame through a same target line could also have conflicts. This is mainly

due to conflicting coda-paths after KLPG test was performed for a target line. These conflicts are between necessary assignments (NAs) in a different time frame and coda paths from the first time frame. Under this scenario, we then place the path for that target gate which was generated in the first time frame and was saved initially in order to cover all the paths.

3) In this compaction approach, we are always saving the path through a target line in first time frame. When we compare patterns for a path with the patterns in the pattern pool, if there is no conflict between them, we then combine the patterns with the first compatible pattern in the pattern pool. There were few cases observed when the combined patterns were not justified by SAT, so they had to be placed separately. In such cases, we place the path that were saved in first time frame directly to the pattern pool. These three scenarios are presented in Tables 1, 2 and 3. For example, from Table 1, considering circuit s1423, 83 paths were placed in the first time frame and 3 paths were placed in the second time frame. 3 paths couldn't be sensitized in a different time frame, so the paths for the target gate that were generated in the first time frame itself was placed in the pattern pool. Similarly, for 1 path, after path was regenerated in a different time frame, the path also had conflicts with the patterns in the pattern pool. So, the saved path in first time frame was placed in pattern pool. 2 paths found the compatible patterns in the pattern pool. But, after they were combined with the compatible patterns, they couldn't be justified by SAT. So, each of those paths which were saved in the pattern pool initially were inserted in the pattern pool. Likewise, in benchmark circuit s38584, 1858 paths were generated in first time frame, and 18 paths were generated in second time frame. 12 paths couldn't be

sensitized in a different time frame, 5 paths had conflict when path was regenerated again in different time frames and 4 paths were not justified by SAT when these paths were combined with the compatible pattern. So, each of these paths were placed as a separate pattern. This behavior can clearly be observed for all the benchmark circuits for different FRAMES configuration in Tables 1, 2 and 3.

Figure 34, 35 and 36 present the bar chart showing the distribution of patterns based on conditions like no paths in other time frames, conflicts in other time frames and not being justified by SAT.

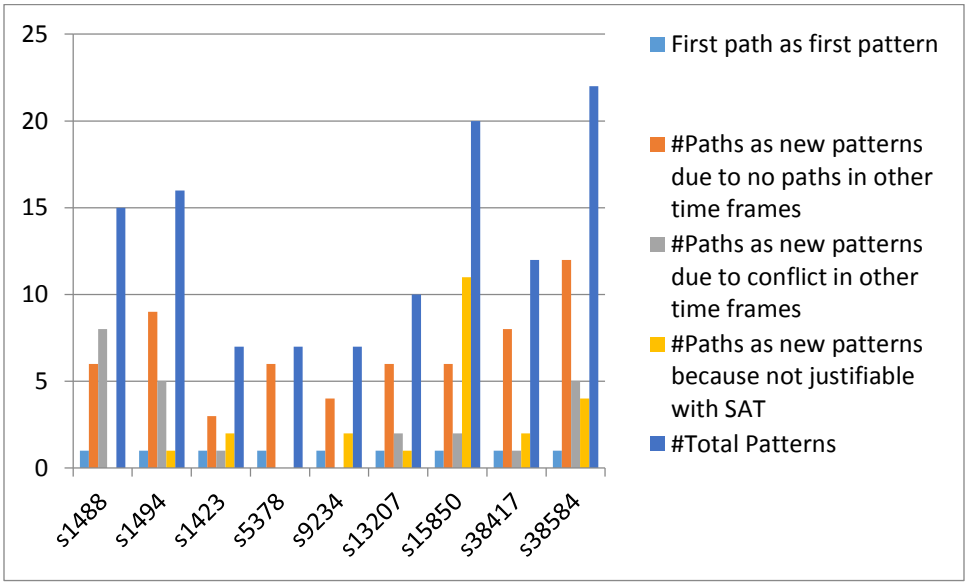

Figure 34. Distribution of patterns, for FRAMES = 3

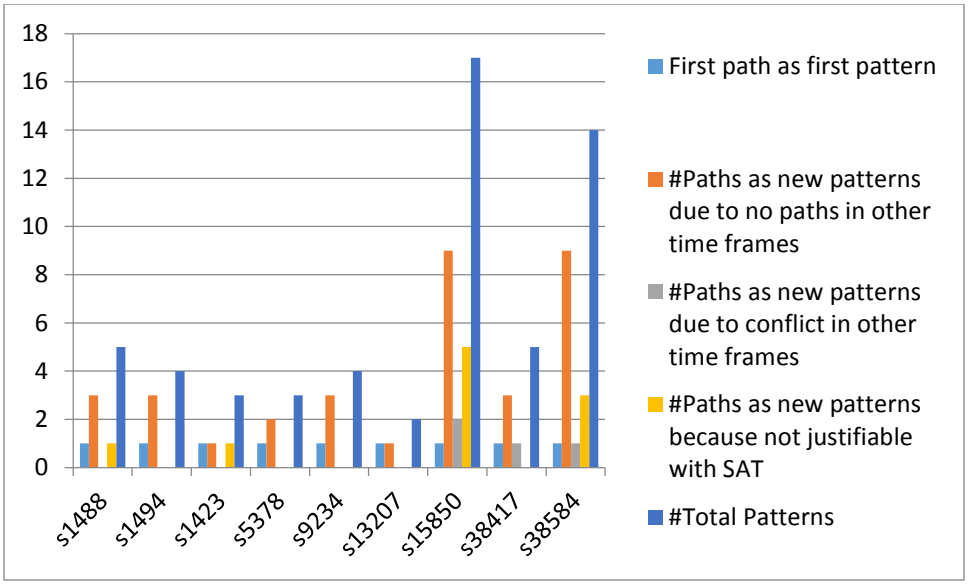

Figure 35. Distribution of patterns, for FRAMES = 4

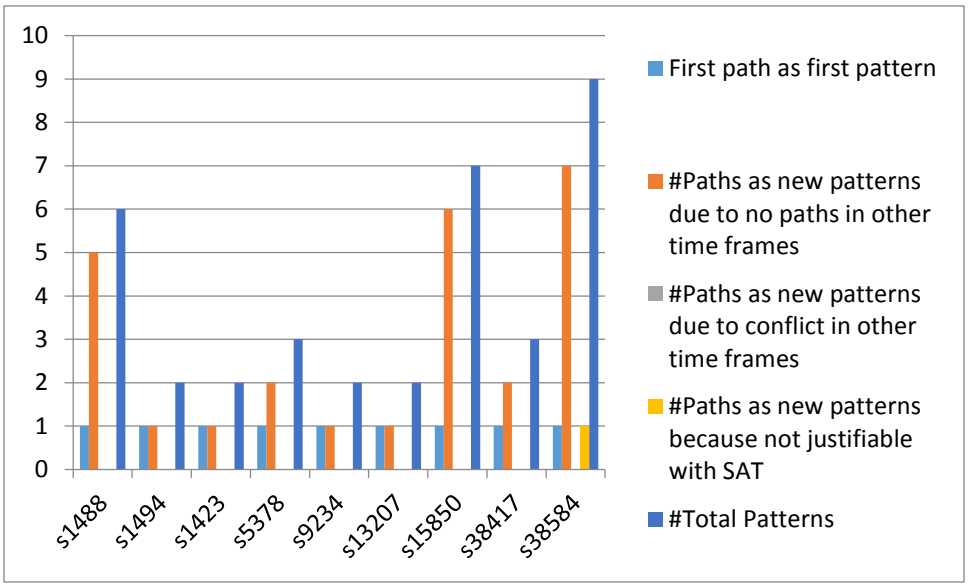

Figure 36. Distribution of patterns, for FRAMES = 5

As we can observe in Figure 34, 35 and 36, when FRAMES = 3, there are cases when new patterns are created when they have conflicts in different time frames compared to FRAMES =4, 5, which was few. New patterns were also created due to not being justified by SAT for FRAMES =3, 4 which was almost rare for FRAMES =5. For FRAMES=5, new patterns are mostly created when paths can't be sensitized in other time frames through the same target line. This could be because the circuit has been unrolled in multiple time frames. Due to this, the sensitization of a path becomes more constrained because it depends upon the values of the preceding frames and the path can't be re-generated in a new time frame resulting in the placement of the original pattern with FRAMES=3 on the pattern pool as a new pattern.

7. CONCLUSIONS AND FUTURE WORK

We have implemented and demonstrated the multi-cycle at-speed approach for compacting patterns and have extended the existing dynamic compaction algorithm. Path to pattern ratio have been increased reducing the number of patterns by some amount. Paths were compacted in time frames following the first time frame if it had conflict with the necessary assignments in the first time frame. It was observed that these paths were few due to the conflicts with the necessary assignments of the coda path. As a part of future work, we could create the coda paths dynamically, in the same way that the SAT engine does the justification through the preamble cycles dynamically. This could improve the results, since it will allow selection of non-conflicting coda paths, allowing more paths to be compacted into a pattern. We could also ignore some unnecessary paths by dynamic allocation of coda paths.

In this approach, if we do not find a path in the first time frame, we ignore that path, because we consider that if a path cannot be found in first time frame, it can also not be found in other consecutive time frames, as the primary inputs are fixed. This can be extended to future work, where the paths can be checked if it exists in other time frames by allowing primary inputs to change. Similarly, in this approach, if there was conflict in the path found in first time frame, we generate K-longest paths through the target in the second time frame. If there is no conflict in second time frame, then we compact the path in the second time frame. There can be a condition that the K-longest path in second time frame could be shorter than the path in first time frame. We are not considering this case

in the research, and taking the path in the second time frame, because we are mainly concerned on compaction of paths into the patterns. This could be extended to future work, where the paths in the other time frames can be exploited which is longer and has no conflicts.

In this research, when a path in first time frame has conflicts with the patterns in the pattern pool, then we try to generate the path through the same target line in different time frames. If there is no path in second time frame, then we are creating a new pattern for the path saved in the first time frame. Future work can be added in this part, to check if the path can be created in third, fourth time frames until a limit has been reached, rather than directly placing the saved path to the pattern pool when the path was not found in second time frame. This could lead to less pattern count.

We performed the experiments in smaller benchmark circuits. We could perform it on larger industrial circuits, and observe the results, as they have a lower care bit density in most patterns, which might provide more space in later time frames for paths to fit. In the future, experiments can also be run on bigger memory arrays, and see if paths could be exploited and compacted in different time frames.

REFERENCES

[1] Wangqi Qiu and D. M. H. Walker, "An efficient algorithm for finding the k longest testable paths through each gate in a combinational circuit," *International Test Conference, 2003. Proceedings. ITC 2003*, pp. 592-601.

[2] W. Qiu *et al*., "K longest paths per gate (KLPG) test generation for scan-based sequential circuits," *2004 International Conference on Test*, 2004, pp. 223-231.

[3] Z. Wang and D. M. H. Walker, "Dynamic Compaction for High Quality Delay Test," *26th IEEE VLSI Test Symposium (vts 2008)*, San Diego, CA, 2008, pp. 243-248.

[4] T. M. Niermann, R. K. Roy, J. H. Patel and J. A. Abraham, "Test compaction for sequential circuits," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 2, pp. 260-267, Feb 1992.

[5] K. O. Boateng, H. Konishi and T. Nakata, "A method of static compaction of test stimuli," *Proceedings 10th Asian Test Symposium*, Kyoto, 2001, pp. 137-142.

[6] Shayak Lahiri, Kun Bian and Duncan Walker "KLPG Based Pseudo Functional Test with Dynamic Compaction," SRC Techcon , 2011.

[7] M. H. Schulz, E. Trischler and T. M. Sarfert, "SOCRATES: a highly efficient automatic test pattern generation system," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 126-137, Jan 1988.

[8] L. N. Reddy, I. Pomeranz and S. M. Reddy, "ROTCO: a reverse order test compaction technique," *Euro ASIC '92, Proceedings.*, Paris, 1992, pp. 189-194.

[9] I. Pomeranz and S. M. Reddy, "On static compaction of test sequences for synchronous sequential circuits," *33rd Design Automation Conference Proceedings, 1996*, Las Vegas, NV, 1996, pp. 215-220.

[10] Ruifeng Guo, I. Pomeranz and S. M. Reddy, "On improving static test compaction for sequential circuits," *VLSI Design 2001. Fourteenth International Conference on VLSI Design*, Bangalore, 2001, pp. 111-116.

[11] J. Wang *et al.*, "Modeling Power Supply Noise in Delay Testing," in *IEEE Design & Test of Computers*, vol. 24, no. 3, pp. 226-234, May-June 2007.

[12] Chakraborty, Avijit (2015). Observability Driven Path Generation for Delay Test. Master's thesis, Texas A & M University.

[13] M. Tehranipoor, K. Peng, and K. Chakrabarty. 2014. "*Test and Diagnosis for Small Delay Defects"*, Springer Publishing Company, Incorporated.

[14] I. Pomeranz, S. M. Reddy, *"Transition Path Delay Faults: A New Path Delay Fault Model for Small and Large Delay Defects"* in Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.16, no.1, pp.98-107, Jan. 2008.

[15] G. L. Smith, *"Model for Delay Faults Based Upon Paths"* in IEEE International Test Conference, Oct. 1985, pp. 342-349.

[16] Wang, Zheng (2010). High Quality Compact Delay Test Generation. Doctoral dissertation, Texas A&M University.

[17] Pokharel, Punj (2013). Path Delay Test Through Memory Arrays. Master's thesis, Texas A & M University.

[18] K. Bian, D. M. H. Walker and S. P. Khatri, "Techniques to Improve the Efficiency of SAT Based Path Delay Test Generation," *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, Mumbai, 2014, pp. 50-55.

[19] S. Chakraborty and D. M. H. Walker, "At-Speed Path Delay Test," *2015 IEEE 24th North Atlantic Test Workshop*, Johnson City, NY, 2015, pp. 39-42.

[20] Laung-Terng Wang, Charles Stroud, Nur Touba "System-on-Chip Test Architectures, 1st Edition Nanometer Design for Testability", Morgan Kaufmann 2010.

[21] Yung-Chieh Lin; Feng Lu; Kai Yang; Kwang-Ting Cheng, "Constraint extraction for pseudo-functional scan-based delay testing," in Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific , vol.1, no., pp.166-171 Vol. 1, 18-21 Jan. 2005

[22] Zhaohui Fu; Yinlei Yu; Malik, S., "Considering circuit observability don't cares in CNF satisfiability," in Design, Automation and Test in Europe, 2005. Proceedings, vol.,no., pp.1108-1113 Vol. 2, 7-11 March 2005

[23] Y. Gao, T. Zhang, D. M. Walker, "Pseudo Functional Path Delay Test through Embedded Memories", Journal of Electronic Testing: Theory and Applications, Volume 31 Issue 1, Pages 35-42, Feb. 2015.

[24] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, "Transition Fault Simulation", IEEE Design and Test of Computers, vol. 4, no. 2, pp. 32-38, Apr. 1987.

[25] L. H. Goldstein and E. L. Thigpen, “SCOAP: Sandia Controllability/Observability Analysis Program”, in Proceedings of ACM/IEEE Design Automation Conference, pp. 190-196, Jun. 1980.

[26] P. Pant, J. Zelman, “Understanding Power Supply Droop During At-Speed Scan Testing,” IEEE VLSI Test Symposium, May 2009, pp. 227-232.